НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Тема курсової роботи:

ПОВЕДІНКОВИЙ ПІДХІД (BDD) ЯК ЕФЕКТИВНИЙ МЕТОД ДЛЯ ОРГАНІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ В БЕЗПЕРЕВНІЙ ПОСТАВЦІ

ІПЗ 121, МП 1

Керівник курсової роботи
д.т.н., доц. Глибовець А.М.

Виконала студентка
Бенюх Л.І..

Київ 2020

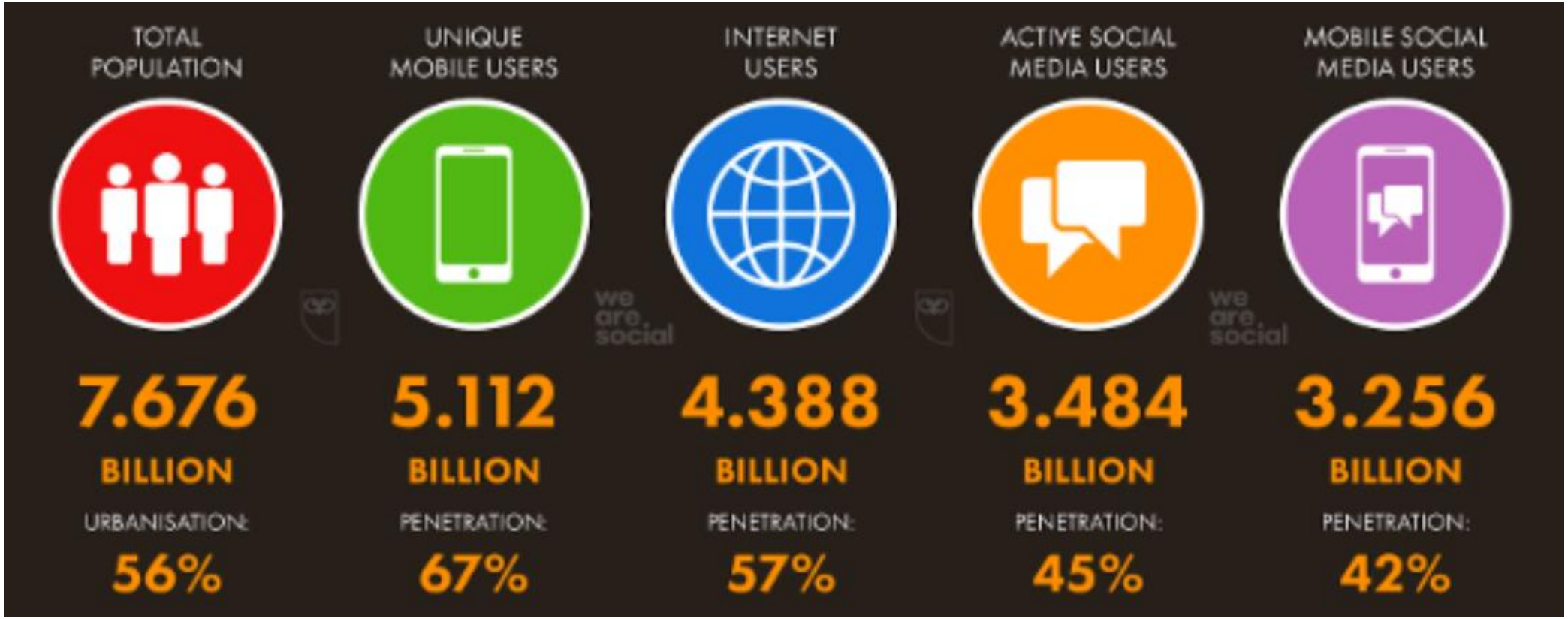
Мета дослідження

Аналіз, реалізація та випробування підходів до автоматизованого тестування у таких умовах як:

- середовище швидких змін
- постійна взаємодія з замовником
- гнучка методологія розробки
- безперервна доставки продукту кінцевому користувачу

Актуальність теми автоматизованого тестування

Статистика по оцифруванню всього світу



https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates

Роль автоматизованого тестування

1. **Безперервна доставка** продукту до користувача
2. **Перевірка системи в цілому** та отримання швидко результатів тестування при поставці кінцевому користувачу
3. **Можливість проходити тест сценарії**, які неможливо виконати вручну одночасно (наприклад паралельні тести, тестування на декількох браузерах та операційних системах)
4. **Швидкість виконання тесту** та отримання результатів тестування
5. **Більш ефективне використання ресурсів** тестування

Рівні тестування у циклі розробки продукту

Покриття тестуванням пов'язане з багатьма діями в рамках програмного забезпечення і життєвого циклу розробки продукту!

Процес побудови автоматизованого тестування

1. Побудова архітектури для автоматизованого тестування
2. Аналіз можливості покриття автоматизованими тестами
3. Створення стратегії по побудові автоматизації тестування на проекті
4. Розробка системи з автоматизованого тестування

Підходи до побудови автоматизованого тестування

1. Розробка на основі тестів (англійською мовою Test Driven Development)
2. Поведінковий підхід (англійською мовою Behaviour Driven Development)
3. Тестування на основі ключових слів (англійською мовою Keyword Driven Testing)
4. Тестування на основі даних (англійською мовою Data Driven Testing)

Переваги поведінкового підходу (BDD):

1. Тести написані мовою, яку розуміють представники бізнес
2. Тестові сценарії можна писати, використовуючи в реальному часі
3. Кожен тестовий сценарій буде дійсним сценарієм для користувача
4. Документацію з тестовими сценаріями можливо буде знайти в одному місці та завжди оновленою
5. Ефективний процес комунікації з клієнтом
6. Прозорість у процесі розробки

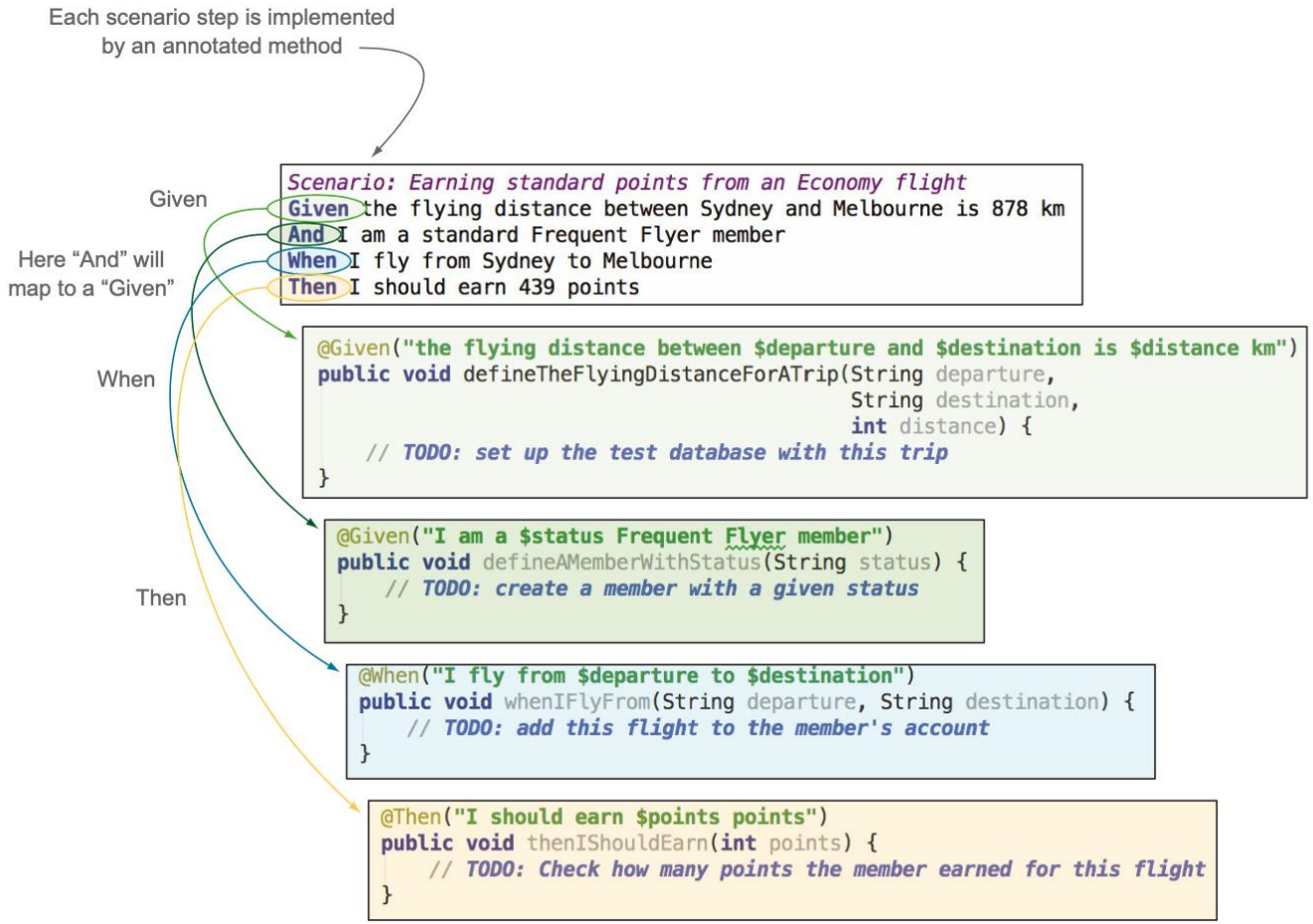
Порівняння існуючих інструментів для BDD

Назва	Жива Документація	Зручність у написанні тестів і документації	Налаштування та робота	Звітування	Інтеграція з CI/CD
Cucumber	Є, і є можливість інтегрувати у CI/CD	Зручно, з стандартами BDD	Зручний у використані, і нескладний у налаштуванні	Так, свій формат. В реальному часі	Так
JBehave	Частково	Зручно, з стандартами BDD	Складний у налаштуванні	Так, але потребує налаштувань	Так
Gauge	Частково	Не дуже зручно(без більш стандартного формату)	Складний у налаштуванні	Так, але потребує налаштувань	Так

Приклад тестового сценарію базуючись на Cucumber + Gherkin

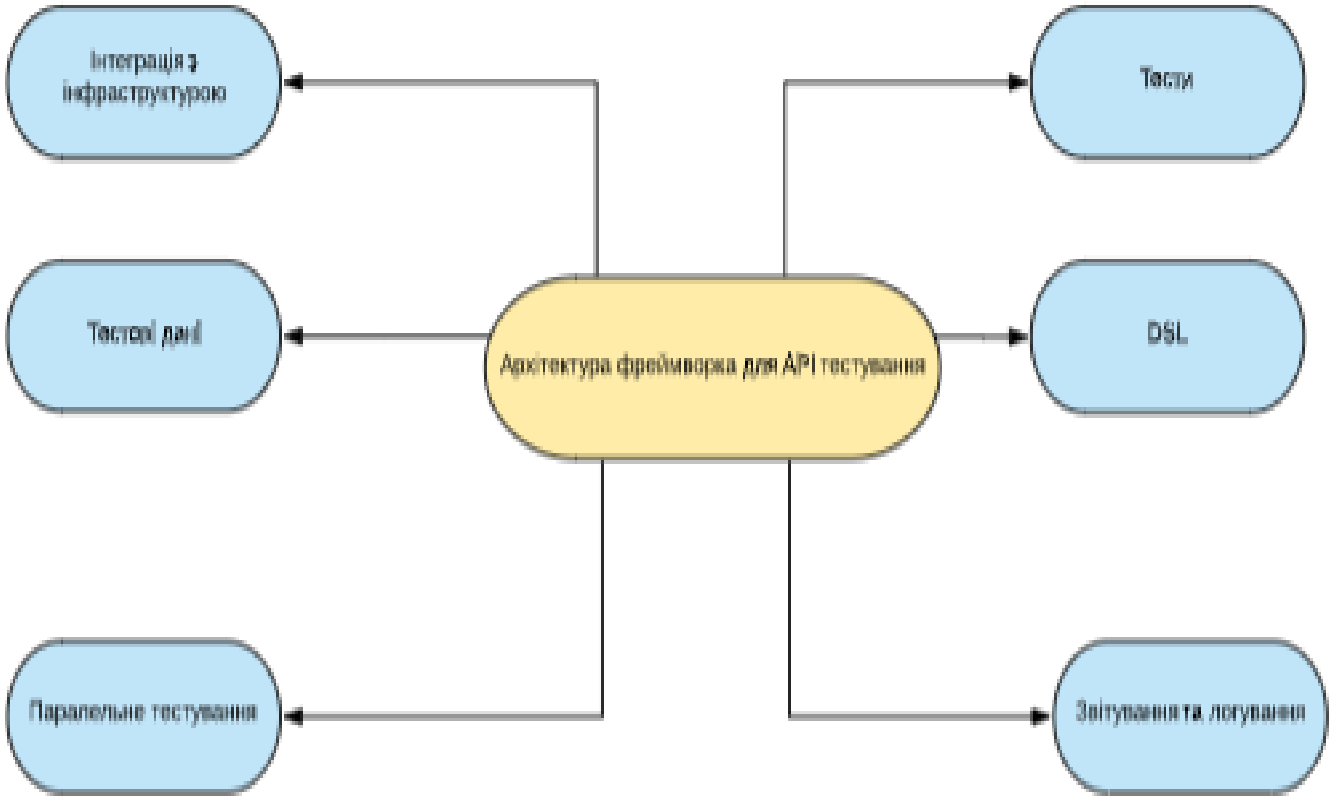
```
Feature: Sign up

  Sign up should be quick and friendly.

  Scenario: Successful sign up

    New users should get a confirmation email and be greeted
    personally by the site once signed in.

    Given I have chosen to sign up
    When I sign up with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message

  Scenario: Duplicate email

    Where someone tries to create an account for an email address
    that already exists.

    Given I have chosen to sign up
    But I enter an email address that has already registered
    Then I should be told that the email is already registered
    And I should be offered the option to recover my password
```

Написання коду до тестового сценарію Cucumber + Java

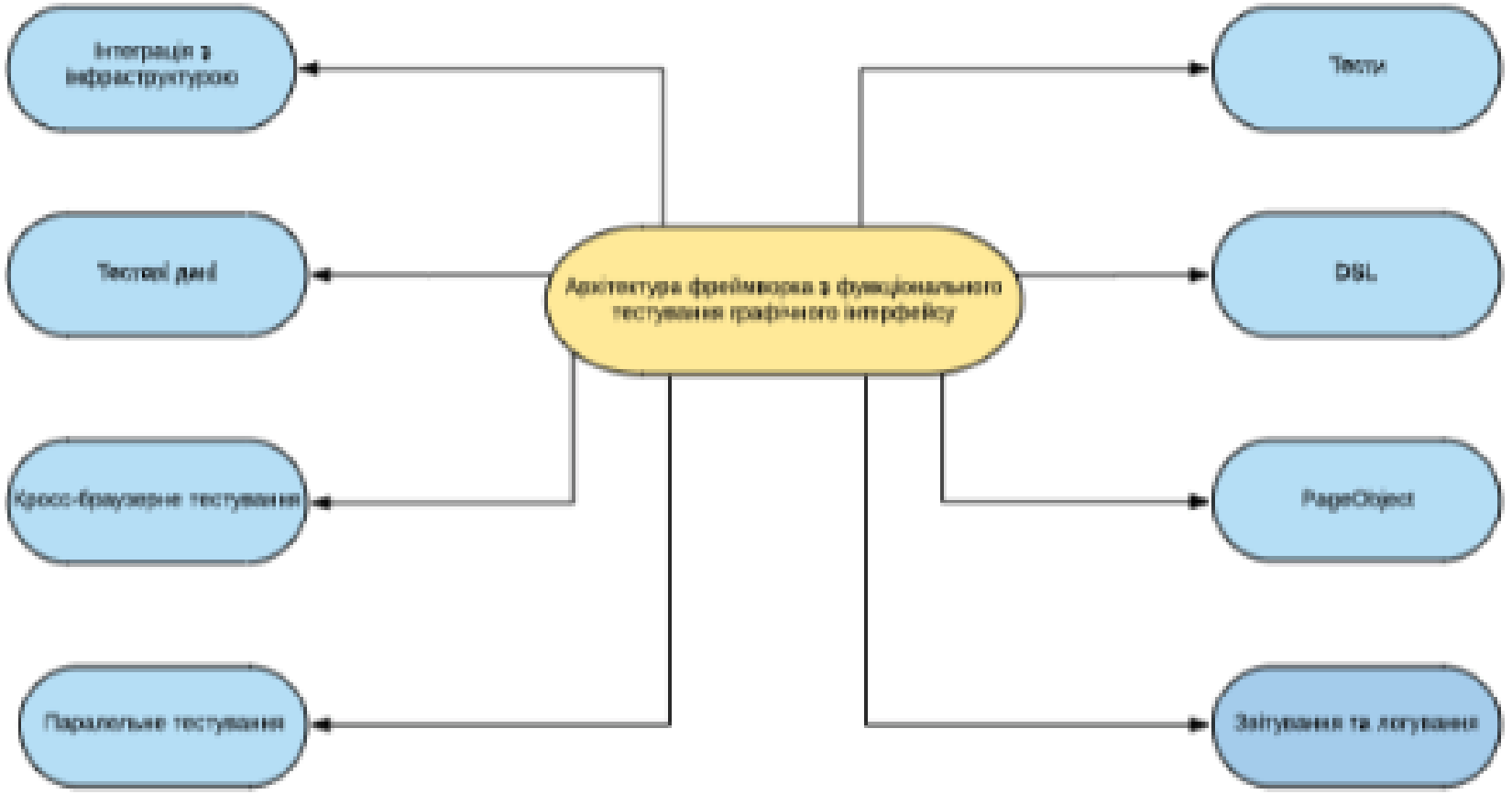


Процес роботи з Cucumber

Компоненти архітектури фреймворка (API тестування)

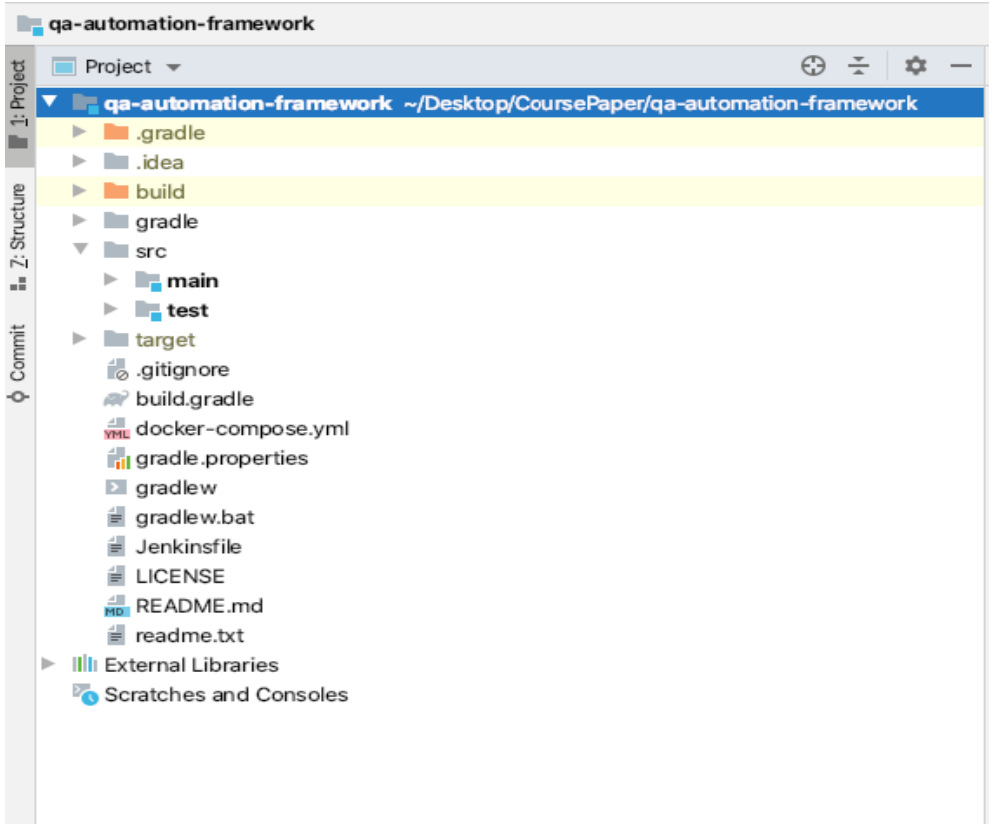


Компоненти архітектури фреймворка (графічного інтерфейсу)

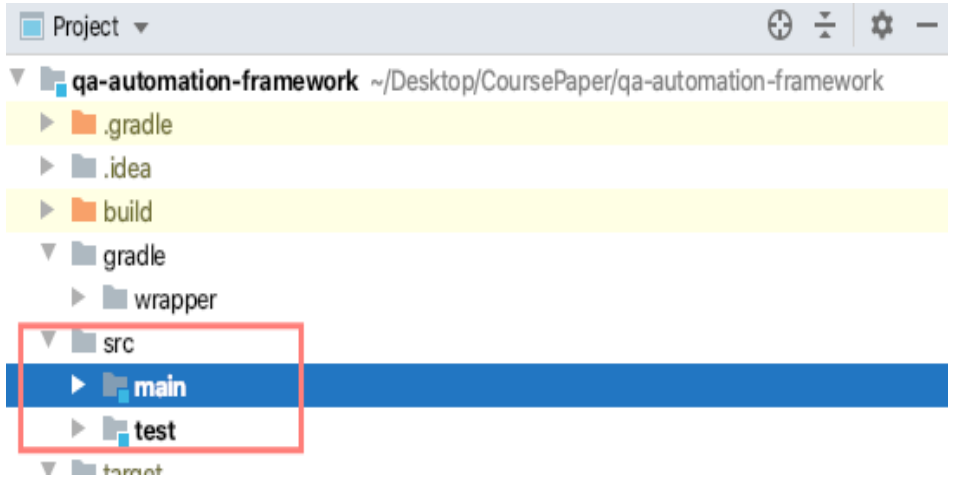


Опис фреймворку

1. Проект

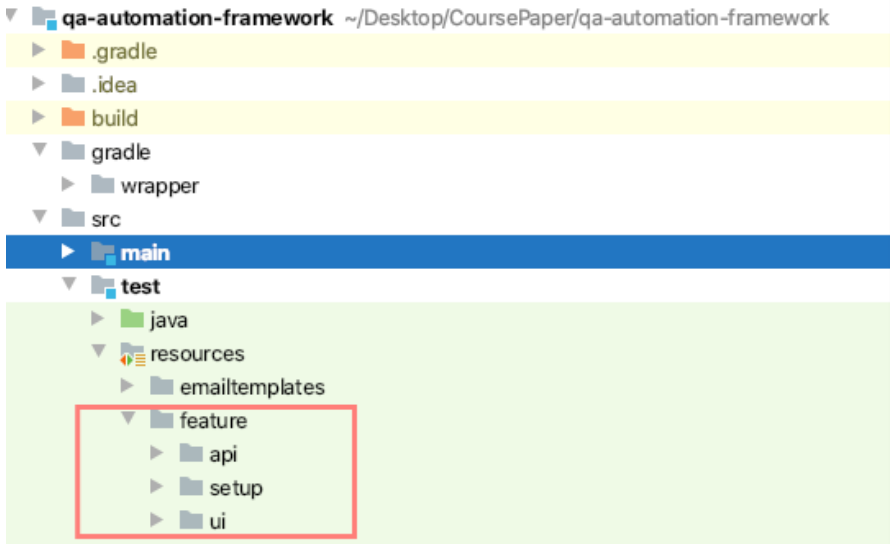


2. Папка src , де буде зберігатися логіка



Опис фреймворку (API тестування)

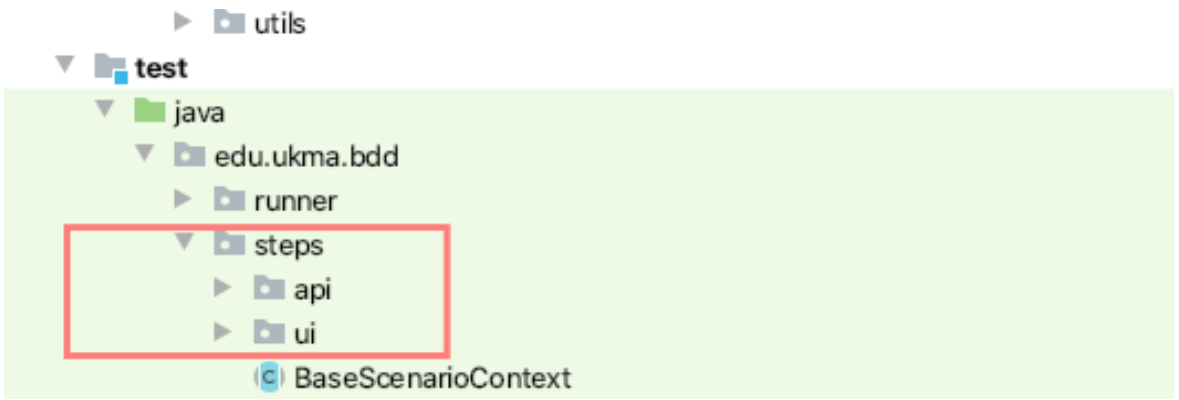
У папці main є resources є папка, яка вміщує feature файл




```
Feature: Test Weather API.

Background:
    Given API baseUrl "https://restapi.demoqa.com/utilities/weather/city/"

@api @api-001 @smoke
Scenario Outline:
    Given City of "<city>"
    And invoke weather API
    Then verify the response code is "<code>" and body contains city name "<verifyCity>"
    Examples:
        | city        | code | verifyCity |
        | kyiv        | 200  | Kyiv       |
        | lviv        | 200  | Lviv       |
        | incorrect   | 400  |            |
```

Опис фреймворку

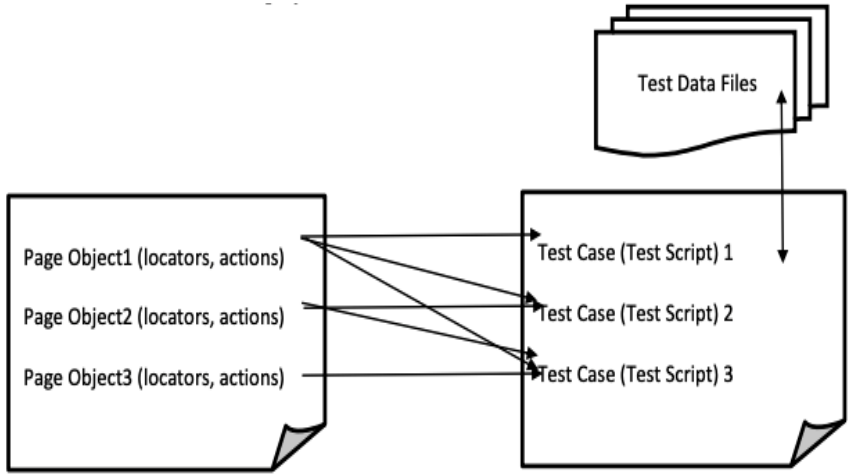
У папці test є папка, яка вміщує код до тестів

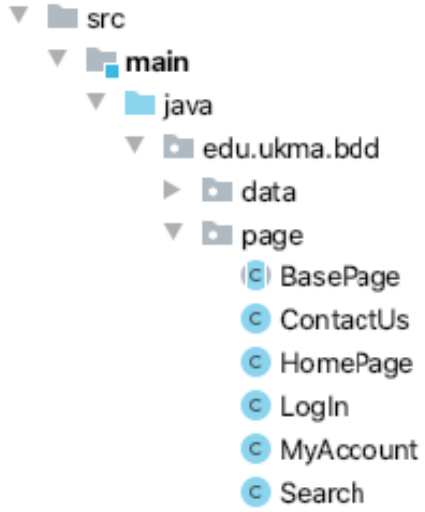



```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;

import io.restassured.response.Response;
import io.restassured.specification.RequestSpecification;

public class APITest {

  private String givenCity;
  private RequestSpecification request;
  private Response response;
  private String baseUrl;

  public APITest() {

  }

  @Given("API baseUrl \"([^\"]*)\"$")
  public void api_url(String baseUrl) { this.baseUrl = baseUrl; }

  @Given("^City of \"([^\"]*)\"$")
  public void city_of(String city) { this.givenCity = city; }

  @Given("^invoke weather API$")
  public void invoke_weather_api() {
    this.request = given().pathParam( parameterName: "city",givenCity);
    this.response = this.request.when().
                        get( path: this.baseUrl+ "{city}");
  }

  @Then("^verify the response code is \"([^\"]*)\" and body contains city name \"([^\"]*)\"$")
  public void verify_response(int code, String city) {
    this.response.then().
        assertThat().
        statusCode(code).
        body( path: "City", city.trim().isEmpty() ? Matchers.nullValue() : Matchers.equalTo(city));
  }

```

Опис фреймворку (UI тестування)

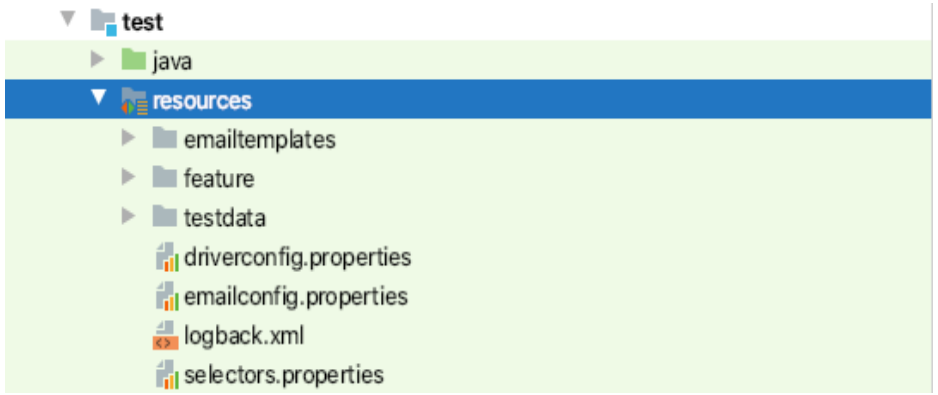
Фреймворк для автоматизованого тестування створений за допомогою PageObject підходу і знаходиться у main папці





Опис фреймворку

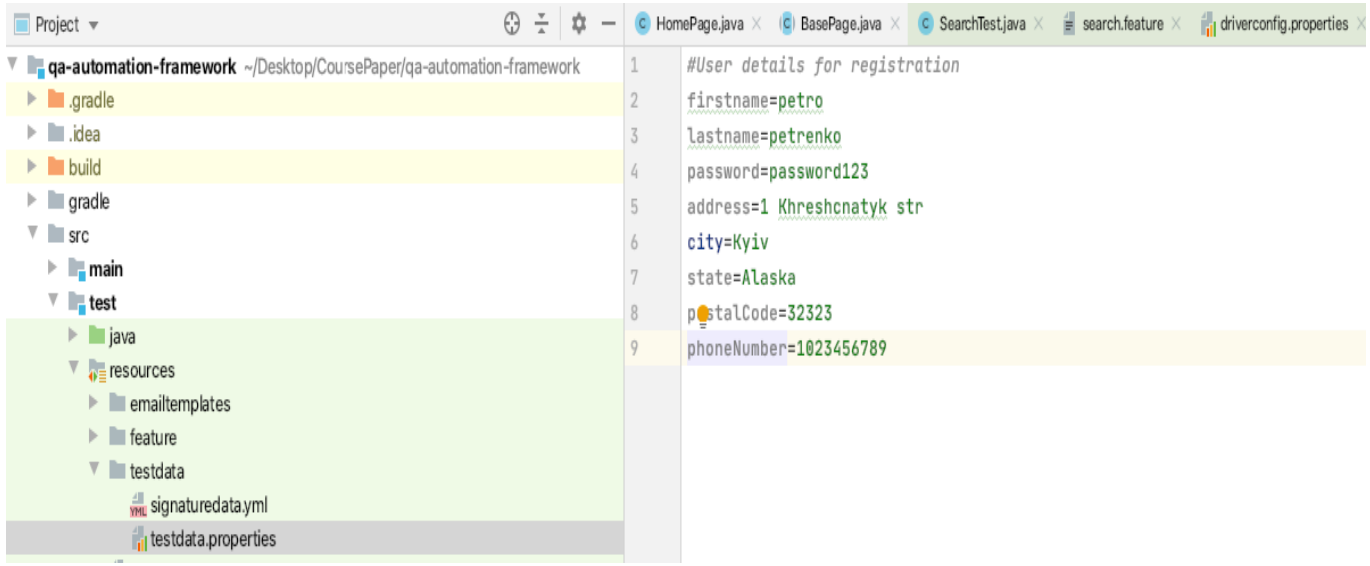
Визначення селекторів за допомогою, яких відбувається пошук елементів на сторонці, можна знайти у файлі selectors.properties




```
#Automation Practice Home Page
contactUsTabCSS=#contact-link > a
signInTabCSS=#header > div.nav > div > div > nav > div.header_user_info > a
searchBarCSS=#search_query_top
searchButtonCSS=#searchbox > button

#Automation Practice Sign In Page
emailAddressCSS=#email
passwordCSS=#passwd
signInButtonCSS=#SubmitLogin > span
authenticationFailedAlertMessageCSS=#center_column > div.alert.alert-danger > p
registerEmailAddressCSS=#email_create
createAccountButtonCSS=#SubmitCreate
titleChoiceCSS=#account-creation_form > div:nth-child(1) > div.clearfix > div:nth-child(3) > label
firstnameCSS=#customer_firstname
lastnameCSS=#customer_lastname
addressFirstnameCSS=#firstname
addressLastnameCSS=#lastname
addressCSS=#address1
cityCSS=#city
stateSelectCSS=#id_state
postalCodeCSS=#postcode
mobileNumberCSS=#phone_mobile
registerButtonCSS=#submitAccount

#Automation Practice Search result page
searchResultListCSS=#center_column > ul > li > div > div.right-block > h5 > a
```

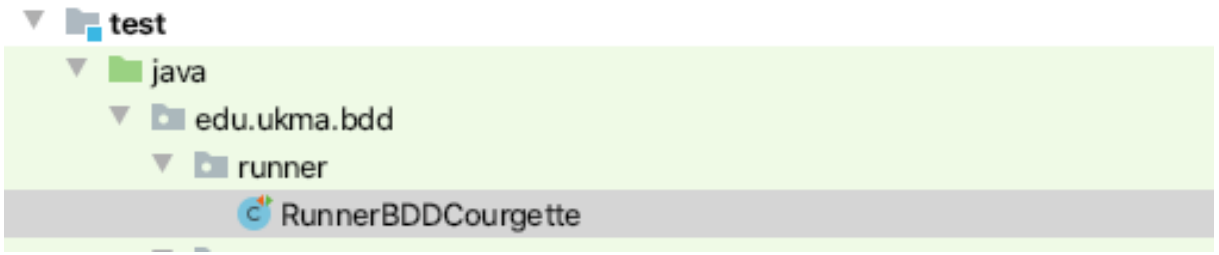
Опис фреймворку

Визначення тестових даних testdata.properties



Огляд класу RunnerBDDCourge

У папці test є клас RunnerBDDCourgette



1. Конфігурація логіки за якою потрібно запускати автоматизовані тести і включає такі параметри як:
2. Кількість потоків для виконання тестів
3. Потрібно чи не потрібно показувати результати тестів
4. Потрібно чи не потрібно пройти заново тест, який не працює тощо

Огляд класу RunnerBDDCourge

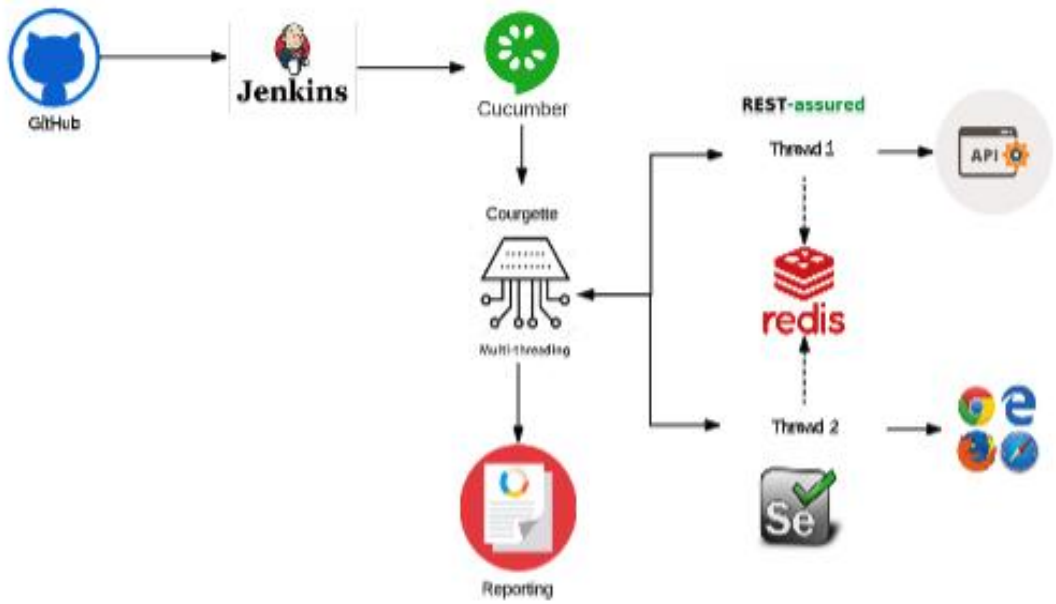
```java
package edu.ukma.bdd.utils;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.Transaction;

public final class RedisUtility {

    private final static Logger LOGGER = LoggerFactory.getLogger(RedisUtility.class);

    private static JedisPool pool;
    private static Jedis jedis;

    private static boolean isRedisAlive;

    private RedisUtility() {
    }

    static {
        configureJedisPool();
    }

    private static void configureJedisPool() {
        try {
            LOGGER.info("Configuring jedis pool");
            pool = new JedisPool(System.getProperty("env.redisHost"), Constants.REDIS_PORT);
            jedis = pool.getResource();
            isRedisAlive = true;
        } catch (Exception e) {
            LOGGER.error("Failed to configure redis", e);
            isRedisAlive = false;
        }
    }

    public static String acquireUser(String userKey) {
        if (!isRedisAlive) {
            String userName = Constants.FALLBACK_USER;
            LOGGER.warn("Returning fallback user {} as jedis is not configured", userName);
            return userName;
        }
```

Огляд класу DriverUtility (UI тестування)

DriverUtility описується з якими браузерами та яким чином можливо взаємодіяти


```
93          }else {
94              LOGGER.warn("No Mode selected for browser");
95              webDriver = new ChromeDriver();
96          }
97          return webDriver;
98      }
99
100     private static WebDriver initFirefoxDriver() {
101         WebDriverManager.firefoxdriver().setup();
102         String mode = System.getProperty("env.mode");
103         FirefoxOptions options = null;
104         WebDriver webDriver = null;
105         if (mode != null) {
106             switch (Constants.Mode.valueOf(mode.toUpperCase())) {
107             case NORMAL:
108                 webDriver = new FirefoxDriver();
109                 break;
110             case HEADLESS:
111                 options = new FirefoxOptions();
112                 options.addArguments("--headless");
113                 options.addArguments("--start-maximized");
114                 webDriver = new FirefoxDriver(options);
115                 break;
116             case INCOGNITO:
117                 options = new FirefoxOptions();
118                 options.addArguments("-private");
119                 webDriver = new FirefoxDriver(options);
120                 webDriver.manage().window().maximize();
121                 break;
122             default:
123                 webDriver = new FirefoxDriver();
124                 break;
125             }
126         }else {
127             LOGGER.warn("No Mode selected for browser");
128             webDriver = new FirefoxDriver();
129         }
130         return webDriver;
131     }
132
133 }
134
```

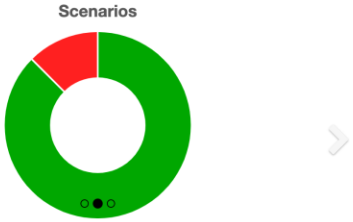
Jenkins pipeline - Архітектура взаємодії компонентів



Візуалізація результатів тестування за допомогою Jenkins та Cucumber системи звітування

Features Statistics

The following graphs show passing and failing statistics for features



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Feature to test searching scenarios	4	0	0	0	0	4	1	0	1	4.684	Passed
Feature to test sign in scenarios	25	1	0	0	0	26	3	1	4	36.592	Failed
Test Weather API.	12	0	0	0	0	12	3	0	3	5.903	Passed
	41	1	0	0	0	42	7	1	8	47.179	3
	97.62%	2.38%	0.00%	0.00%	0.00%		87.50%	12.50%			66.67%

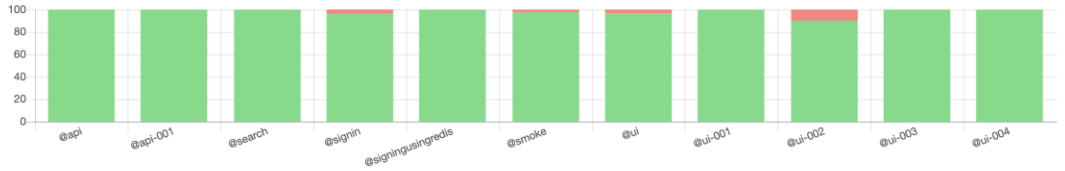
Steps Statistics

The following graph shows step statistics for this build. Below list is based on results. step does not provide information about result then is not listed below. Additionally @Before and @After are not counted because they are part of the scenarios, not steps.

Implementation	Occurrences	Average duration	Max duration	Total durations	Ratio
edu.ukma.bdd.steps.api.APITest.api_url(java.lang.String)	3	0.001	0.001	0.003	100.00%
edu.ukma.bdd.steps.api.APITest.city_of(java.lang.String)	3	0.003	0.004	0.010	100.00%
edu.ukma.bdd.steps.api.APITest.invoke_weather_api()	3	1.706	2.886	5.119	100.00%
edu.ukma.bdd.steps.api.APITest.verify_response(int,java.lang.String)	3	0.257	0.716	0.771	100.00%
edu.ukma.bdd.steps.ui.SearchTest.user_is_browsing_Automation_Practice_website()	5	0.024	0.064	0.123	100.00%
edu.ukma.bdd.steps.ui.SearchTest.user_lands_on(java.lang.String)	11	0.006	0.009	0.071	90.91%
edu.ukma.bdd.steps.ui.SearchTest.user_on_on_on_ecom(java.lang.String,java.lang.String)	4	2.600	3.190	10.400	100.00%
edu.ukma.bdd.steps.ui.SearchTest.user_search_for(java.lang.String)	3	3.873	4.550	11.621	100.00%
edu.ukma.bdd.steps.ui.SearchTest.verifies_all_result_product_contain_the_word(java.lang.String)	3	0.070	0.094	0.210	100.00%
edu.ukma.bdd.steps.ui.SignInTest.inputs_email_address_and_password()	2	4.990	5.377	9.981	100.00%
edu.ukma.bdd.steps.ui.SignInTest.inputs_email_address_and_password(java.lang.String,java.lang.String)	2	4.435	4.614	8.870	100.00%
edu.ukma.bdd.steps.ui.UIScenarioContext.endMethod(io.cucumber.java.Scenario)	5	0.201	0.471	1.009	100.00%
edu.ukma.bdd.steps.ui.UIScenarioContext.startMethod(io.cucumber.java.Scenario)	5	5.517	6.405	27.585	100.00%
13	52	1.457	27.585	1:15.773	Totals

Tags Statistics

The following graph shows passing and failing statistics for tags

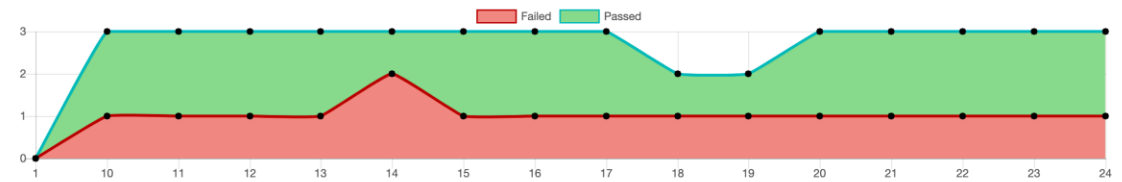


	Steps						Scenarios			Features	
Tag	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
@api	9	0	0	0	0	9	3	0	3	5.900	Passed
@api-001	9	0	0	0	0	9	3	0	3	5.900	Passed
@search	4	0	0	0	0	4	1	0	1	4.684	Passed
@signin	25	1	0	0	0	26	3	1	4	36.592	Failed
@signingusingredis	16	0	0	0	0	16	2	0	2	22.689	Passed
@smoke	38	1	0	0	0	39	7	1	8	47.176	Failed
@ui	29	1	0	0	0	30	4	1	5	41.276	Failed
@ui-001	4	0	0	0	0	4	1	0	1	4.684	Passed

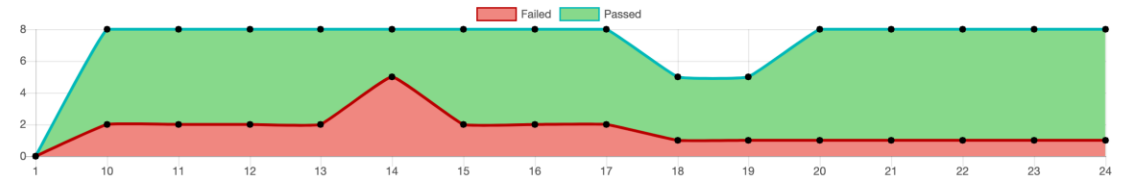
Trends Statistics

The following graph shows features, scenarios and steps for a period of time.

Features:



Scenarios:



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Дякую